NYU TANDON SCHOOL OF ENGINEERING Name: _____

ECE648 – Real Time Embedded Systems ID: _____

Midterm 1 Spring 2025

Instructions: Answer all 3 questions. You may use any reference materials from class or from our class websites and/or reference sheets. Collaboration is not permitted.

Question 1:

We spent a significant amount of time in class discussing how to read and write memory using both C and assembly. Since our architecture uses memory mapped I/O, setting up the GPIO registers is simply reading and writing the appropriate memory locations. The following algorithm (A,B,C,D) can be used for our controller to do a block GPIO write, meaning writing all port pins (15 in total) simultaneously, for any port (say PORTA).

PORTA has a base address of 0x40020000

A. Set each pin in the MODER register to general purpose output mode

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

B. Set each pin in the OTYPER register to output push-pull

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the output type of the I/O port.
0: Output push-pull (reset state)
1: Output open-drain

C. Set each pin in the PUPDR register to 0, meaning no pull up or down resistor

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDRy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O pull-up or pull-down
00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

D. Set the ODR register bits to a 1 or 0, depending on if the pin is turned on or off.

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)
These bits can be read and written by software.
Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..I/J/K).

a. Write a C function:

```
uint32_t  BlockWritePortA(uint16_t PinVals, uint16_t PullUP) {

//Code Here
}
```

PinVals is a 16 bit number with 1's in positions to turn "on" and 0's in the pin positions to turn "off". PullUP is a 16 bit mask, where the bit positions with a "1" will have a pull-up resistor enabled. Positions with a "0" will have no pull-up enabled. The function should implement algorithms parts A., B., C. and D. described above and return the value of the ODR register above. Note, the addresses of the registers (memory) are the Base Address + Offset as indicated above.

b. Write an ARM assembly function called "_BlockWritePortA" that implements part a. above.

NOTE: Be careful how you use LDR, as there are limits to the "mem" parameter!

```
.globl _BlockWritePortA

.syntax unified


_BlockWritePortA

;

; Code Here

;

bx lr
```

c. **Write the short C program that calls the ARM Assembly function _BlockWritePortA in its main();**

Question 2:

Consider the following ARM assembly code segment.

a. Accurately comment each line of code
b. Describe what parameters r0 and r1 (passed into the function) are used for.
c. What are each of the local variables r4-r8 used for?
d. What is the purpose of this function?
e. Explain in detail the specific purpose of “stmfd” and “ldmfd” in this function.

```
.globl _MyFunc
.text
_MyFunc:
 stmfd sp!, {r4, r5, r6, r7, r8, lr}
 cmp r1, #1
 ble end_outer

 sub r5, r1, #1
 mov r4, r0
 mov r6, #0

loop_start:
 ldr r7, [r4], 4
 ldr r8, [r4]
 cmp r7, r8
 ble no_go

 mov r6, #1
 sub r4, r4, 4
 swp r8, r8, [r4]
 str r8, [r4, 4]!
no_go:
 subs r5, r5, #1
 bne loop_start

end_inner:
 cmp r6, #0
 beq end_outer

 mov r6, #0
 mov r4, r0
 sub r5, r1, #1
 b loop_start

end_outer:
        ldmfd   sp!, {r4, r5, r6, r7, r8, pc}
```

Question 3:

Suppose you want to use your microcontroller to perform complex math, like a DFT for example, which requires a way to store and manipulate complex numbers.

a. Design a global array variable that is capable of storing 20 complex numbers in terms of their real and imaginary parts. You can use any data types, including structs, to store the array of 20 complex numbers.
b. Write a C function called:

```
retrieveValue(x,n)

//returns a pointer to the "n th" complex number in the array
//stored at address x
```

c. Write the same function in ARM assembly.